

# Sangam – A Distributed Pair Programming Plug-in for Eclipse

Chih-Wei Ho,<sup>1</sup> Somik Raha,<sup>2</sup> Edward Gehringer,<sup>1</sup> Laurie Williams<sup>1</sup>

<sup>1</sup>Department of Computer Science, North Carolina State University  
Raleigh, NC 27606, USA, {cho, efg, lawilli3}@ncsu.edu

<sup>2</sup>Industrial Logic  
Berkeley, CA 94708, USA, somik@industriallogic.com

## Abstract

In pair programming, two programmers traditionally work side-by-side at one computer. However, in globally distributed organizations, long-distance collaboration is frequently necessary. Sangam is an Eclipse plug-in that allows Eclipse users in different locations to share a workspace so that they may work as if they were using the same computer. In this paper, we discuss the Sangam plug-in, and our experience developing it via distributed and collocated pair programming.

## 1 Introduction

Due to the popularity of Extreme Programming (XP) [2] and the demonstrated benefits of pair programming (PP) [3], programmers are increasingly pairing with their colleagues during software development. Traditionally, PP requires programmers to work on the same computer. Although research shows that collaboration at a distance is difficult to achieve [5], globalization of industry and increasingly distributed workforces are causing programmers on the same team increasingly to work at different sites, even on different continents. To enjoy the benefit of PP, distributed team members need to work as if they were sitting next to each other. In terms of productivity and quality, distributed pair programming can be as effective as collocated PP [1].

We have developed Sangam<sup>1</sup> (which means “confluence” in Hindi, from the Sanskrit *samagama*), a plug-in for Eclipse, as a tool for distributed PP (DPP). Sangam provides a user interface specifically for DPP, and synchronizes the development environments for both programmers. The plug-in itself is being developed by a distributed team led by the authors of this paper. This paper, as well as discussing the implementation of Sangam, will discuss available tools that support distributed collaboration and our experience with DPP.

The rest of this paper is organized as follows. Section 2 provides the background of PP and distributed collaboration. Section 3 describes our experience of collocated and distributed PP. Section 4 presents the design of Sangam. Section 5 gives the current status of our work and suggests future directions.

## 2 Background

PP has been practiced sporadically for decades [6], but the popularity of the practice has grown by its use within the XP [2] software development methodology. This section provides background information on PP and distributed collaboration. Early evidence for the effectiveness of PP was only anecdotal. Recent research shows that pair programmers produce code with higher quality without sacrificing productivity, and have greater

---

<sup>1</sup> <http://sangam.sourceforge.net>

<sup>3</sup> <http://www.sourceforge.net/>

job satisfaction [3] and confidence in their products [7]. To enjoy the benefits of PP when it is not possible to work at one computer, we need a virtual collaboration environment in which distributed developers can work as if they were sitting together, using the same input devices and looking at the same monitor.

Collaboration may be either synchronous or asynchronous. With asynchronous collaboration, the team members work at different times and integrate their work after they finish their tasks. There is typically a centralized server where the communication takes place for collaborative programming efforts. SourceForge<sup>3</sup> is one example of such a server. PP belongs to the other category, synchronous collaboration.

There are two basic approaches for distributed synchronous collaboration [4] via the Internet. The first one is to broadcast the display of any application from a member to all the others. This approach requires a large amount of image information to be transmitted across the network, but all applications can be used in the environment without any modification. Some applications, including VNC<sup>4</sup> and Microsoft NetMeeting,<sup>5</sup> use this method to share one person's desktop with collaborators. Baheti et al [1] used NetMeeting for the DPP experiments mentioned earlier.

The second approach is to design an application specifically for distributed collaboration. This allows the user interface to be more attuned toward collaboration. One example application is Yahoo! Messenger.<sup>6</sup> In Messenger, when a user sends a text message, the text, rather than the screen buffer of the chat window, is sent through the network. It has a user interface designed for composing text messages.

Our experience shows that, while shared-desktop tools can be used for distributed PP, the display refresh rate can be too low for the programmers to understand what their partners are doing, especially when a high-speed network connection is not available. Therefore, we propose a plug-in that is specifically designed for DPP in the Eclipse development environment. Rather than sending screen-buffer information through the network, Sangam transmits messages that are important for PP. The information content is significantly less than for image data,

allowing developers to use this plug-in without a broadband network

## 3 Experience

During the development of Sangam, we gained experience with both collocated and distributed PP. Our team consisted of four North Carolina State University students and four independent programmers in California. When pairing with a non-local partner, we used VNC for desktop-sharing, Eclipse for the development environment, and Yahoo! Messenger for voice communication. Our experience on this project is described in this section.

### 3.1 Collocated vs. Distributed

During this project, we held weekly long-distance conferences for iteration planning and velocity [2] tracking. We discovered that we delivered approximately the same amount of functionality in the same amount of time with collocated pairing as with distributed pairing. Because we were so engaged in programming tasks, we felt satisfied (and exhausted) after each pairing session, whether collocated or distributed. Furthermore, we found PP to be an efficient method for knowledge exchange. Although we did have a Sangam developer's guide,<sup>7</sup> most of the required skills and techniques for this plug-in were exchanged during PP. We believe that this demonstrates the effectiveness of distributed PP.

One difference between collocated and distributed PP is the sense of presence. With DPP, the navigator could lose concentration more easily because the voice of the driver came from the loudspeakers, not from someone sitting next to the navigator. However, this was not a problem for us. When the navigator was not paying attention, the driver would notice that the navigator was talking less and ask the navigator to contribute something. When working collocated, it was easier to stay focused.

Another difference is in the social aspect of programming in pairs. When the project started, the developers did not know each other. The collocated developers became friends after first few pairing sessions. However, there was still a sense of unfamiliarity with the non-local developers,

---

<sup>4</sup> <http://www.realvnc.com/>

<sup>5</sup> <http://www.microsoft.com/windows/netmeeting/>

<sup>6</sup> <http://messenger.yahoo.com/>

---

<sup>7</sup> <http://sangam.sourceforge.net/SangamDocumentation.html>

despite the fact that they programmed very effectively together.

## 3.2 Tool Support

Our development team was distributed from coast to coast. Most of the time, we had a broadband Internet connection. The shared-desktop tool we used worked well. Nevertheless, we did notice the following shortcomings with the tools we used that made them suboptimal for PP:

1. Unlike collocated pair programmers, distributed pair programmers both have control of their mice and keyboards, and can move the mouse or type a keystroke at any time. It is irritating to the driver when the navigator hits the keyboard or moves the mouse accidentally.
2. A low display-refresh rate can sometimes be confusing. Something significant may be lost at the remote display. For example, if the driver copies some text from an editor and pastes it into another editor, the navigator may only see the new content of the latter editor, without knowing where it came from.
3. It is better if both developers use the same resolution for their monitors. Otherwise, one may lose the trace of the mouse pointer.

These problems can be addressed with tools that are specifically crafted for DPP, such as Sangam.

## 4 Plug-in Design

We use an event-driven design for this plug-in. When the driver does something in Eclipse (e. g. cut and paste some code), the plug-in intercepts the event and notifies the plug-in at the navigator's end to automatically perform the same task. There are three basic components in our design: event interceptor, message server, and event reproducer. Following are the detailed descriptions of these components.

The responsibility of the event interceptor is to capture the event when the driver does something in Eclipse, and then send it to the message server. Since Eclipse is an open environment, a user may install a virtually unlimited number of plug-ins. It is impossible to capture all the events generated by every plug-in. Our goal is to write a plug-in to support synchronous development in Java. Therefore, we only focus on Java editor

events, program-launching events, and resource-change events.

We use a centralized server for message handling. All developers who want to participate in a programming session need to connect to the same server. More than two developers may join a programming session, but only one can drive at a time. We use Kizna SyncShare<sup>8</sup> as the message server because of its lightweight protocol and SDK for easy development. Though the server can run on an individual machine, we developed an Eclipse plug-in for SyncShare so that the server can also run within the development environment.

When the driver does something in Eclipse, the action needs to be reproduced at the navigator's computer. The *event reproducer* does this. When it receives a message from the message server, it parses the message and interacts with Eclipse to perform the driver's action on the navigator's machine. This allows the navigator to see whatever the driver is doing in Eclipse.

Figure 1 on the next page shows an Eclipse workspace after installing Sangam. The Sangam Editor provides functionality for the developers to edit source code synchronously. The Sangam Launchers enable the developers to launch a Java application or JUnit test together. The developers use the Sangam Toolbar to connect to or disconnect from the message server (the left button on the toolbar) and Start/Stop Driving (the right button on the toolbar). In our implementation, a developer becomes the driver by pressing the Start Driving button. The driver then has the control of mouse and keyboard until he or she hits the Stop Driving button. The navigator may also move the mouse, but this does not affect the cursor on the driver's screen. Currently this plug-in does not prevent the navigator from typing, so pair programmers need to work out a protocol that one should not use the keyboard unless driving.

---

<sup>8</sup> [http://www.kizna.com/products\\_sync.html](http://www.kizna.com/products_sync.html)

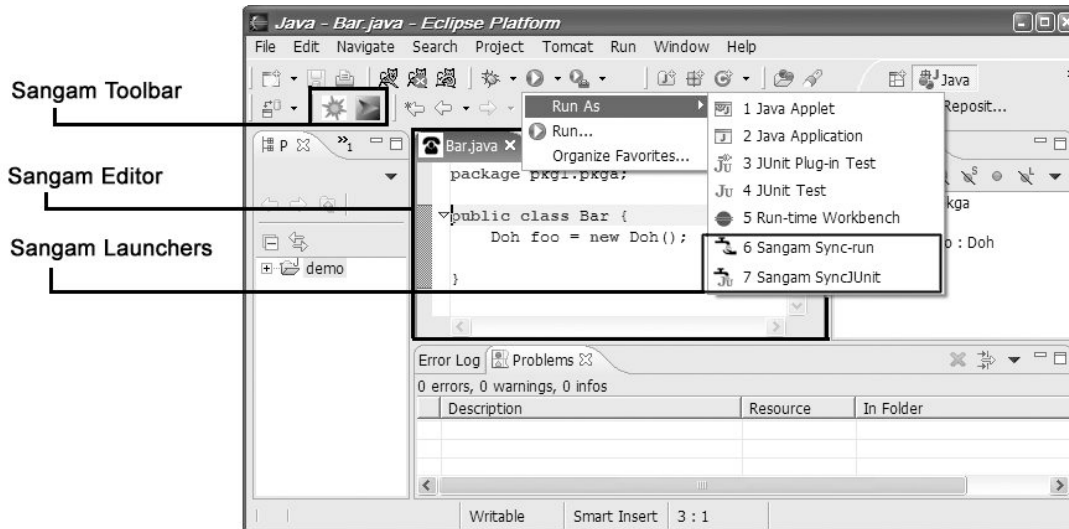


Figure 1: Sangam Plug-in

We think it is important for the navigator to visualize all actions of the driver. In our design, the event reproducer uses the Eclipse API to perform all of the driver's actions on the navigator's screen. The actual layout of the Eclipse window does not matter at all.

Currently, the lack of extensibility is the greatest limitation of this plug-in. Because the tool can only catch and reproduce predefined events, it cannot support PP in other programming languages or editors. For example, this plug-in cannot be used with CDT (C/C++ Development Tools) because it does not know how to intercept CDT events.

## 5 Results and Future Work

The development of Sangam is an ongoing effort. We keep adding new features to make Eclipse a platform for PP. In the current version, Sangam provides the following features:

- Editor synchronization: This includes typing, selection, opening, closing, and viewport scrolling synchronization.
- Launch synchronization: The programmers can launch or debug the same Java application or JUnit test at the same time.
- Resource synchronization: When the driver adds, deletes, or modifies the files on his or her local disk using Eclipse, the same changes will also be reflected in the navigator's local disk.

- Refactoring synchronization: From the perspective of the Eclipse API, refactoring is a complicated set of resource and editor changes. A single refactoring may affect many files and modify the contents of different editor windows. Sangam is designed to deal with some special cases raised by refactoring within Eclipse. For example, the renaming of a class requires renaming of a file. There is no "rename" event in Eclipse; rather this is realized by deleting one file and creating a new one. Therefore, Sangam will transmit the new file to the navigator's computer and delete the old file there. If a change of selection or viewport is sent to a file that has been deleted, the editor will crash. Therefore, the new file must first be opened, and the Eclipse event directed to it.

We try to synchronize the workspaces of the programmers participating a pairing session. The synchronization, however, conflicts with CVS support in Eclipse. After a pairing session, because every programmer has the newest version of the files, these files will be labeled as an outgoing change. Nonetheless, only one programmer can check in the code. The others need to update it from CVS, although the source code is already up to date.

To provide complete support for synchronized distributed collaboration, we plan to continue this project in three different directions:

1. CVS support: This plug-in needs to work with CVS plug-in so that when the driver

commits the source code, the files will be labeled correctly at the navigator's machine.

2. Data collection: The event-driven design makes this plug-in an ideal tool for collecting accurate data in PP, such as the amount of time spent driving or navigating and the amount of time required to pass a JUnit test. The collected data will be used for further research on PP and distributed collaboration.
3. Development activity support: Coding is but one part of software development. Other development activities, like the planning game [2], can also benefit from distributed collaboration. Our ultimate goal is to make Eclipse a collaborative environment for the whole software lifecycle.

## Acknowledgments

The authors would like to thank the distributors of this project for their participation in the development of Sangam. The development of the Sangam was funded in part by an IBM Eclipse Innovation Award.

## About the Authors

Chih-wei Ho is a Ph. D. student in Computer Science at NCSU. His interest is in the agile software process. Somik Raha is the founder of the Sangam project. He is a professional XP coach. Ed Gehringer and Laurie Williams are an associ-

ate professor and assistant professor at NCSU, respectively.

## References

- [1] P. Baheti, E. Gehringer, and D. Scotts. Exploring the efficacy of Distributed Pair Programming. In *Proceedings of Extreme Programming and Agile Methods – XP/Agile Universe 2002*, pages 208–220, Chicago, Illinois, USA, 2002.
- [2] Kent Beck. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.
- [3] A. Cockburn and L. Williams. The Costs and Benefits of Pair Programming. In *Extreme Programming Examined*, pages 223–247, Addison-Wesley, 2001.
- [4] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: Some Issues and Experiences. In *Communications of the ACM*, Vol. 34, Issue 1, Pages 39–58, January 1991.
- [5] G. M. Olson and J. S. Olson. Distance Matters. In *Human-Computer Interaction*, Vol. 15, pages 139–179, 2000.
- [6] L. Williams and R. R. Kessler. *Pair Programming Illuminated*, Addison-Wesley, 2002.
- [7] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the Case of Pair Programming. In *IEEE Software*, Vol. 17 Issue 4, pages 19–25, July/August 2000.

